Towers of Hanoi

rs@toprllc.com June 26, 2020

Introduction

It is not surprising that as the size of a problem - however **the size** is defined, will determine the cost of a solution. For a given problem, there may be multiple solutions possible and we may have to understand the cost before settling on the solution. Facial recognition for example is such a problem where there may be a large database to search for a match. Confronted by a choice, algorithms need to be evaluated before one is chosen for implementation.

In this projectlet, the familiar problem **Towers of Hanoi** is used to explore such ideas. `https://en.wikipedia.org/wiki/Tower_of_Hanoi` defines the basic problem and outlines a **recursive** and an **iterative** solution. This can be implemented with appropriate instrumentation to compare the performance.

Wikipedia article referenced above also points to several extensions e.g. higher number of pegs and other rule set governing the moves.

The goal is to experiment with ideas for instrumentation for performance comparisons. In particular, it is posited that algorithm visualization graphically gives a good insight into the performance of algorithms - in comparison with a simpler text output based comparisons. As we explore visualizations, the separation of the algorithm from the visualization becomes a topic to explore as well. A clean separation will enable substitution of visualization techniques and toolkits without compromising the algorithm.

Initial approach

The two factors that are impacted by the size of the problem are execution time and memory usage. A reasonable assumption might be that both recursive and iterative solutions utilize the same data structure (which may not always be true). Analytically it has been proven that the minimal number of moves for n disks is $2^{(n)} - 1$. Implementation of both the recursive and the iterative algorithms, illustrating the instrumentation techniques demonstrates this.

In the case of a recursive approach, the depth of recursion will be a proxy for space since each invocation requires stack growth.

Results

For a recursive solution, we find that the maximum depth is the same as the number of disks. The total number of moves however is $2^{(n)} - 1$.

```
Total no of moves 65535 Iterations 0 Max Depth 16
```

For an iterative solution, the number of disk moves is still the same ie $2^{(n)} - 1$.

```
Total no of moves 65535 Iterations 21845 Max Depth 0
```

More experimentation leads to the following:

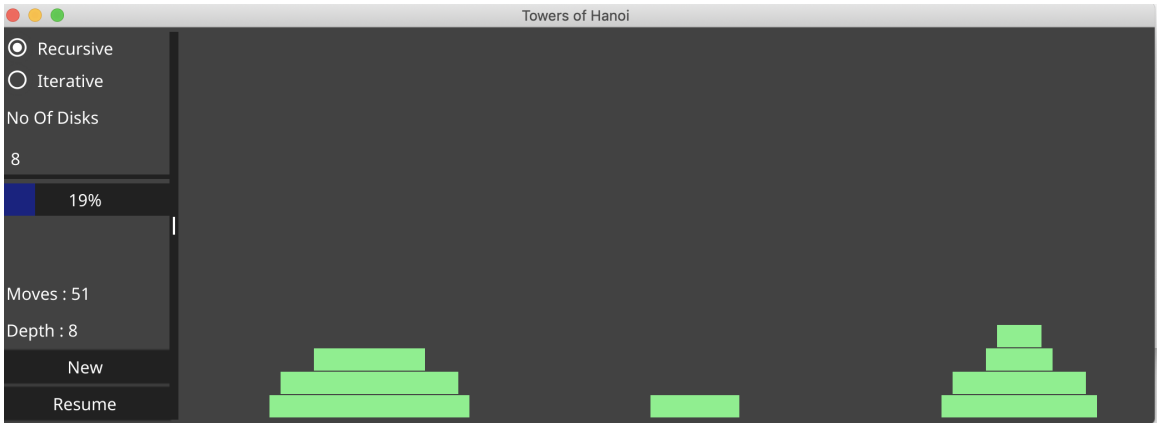
Disks	Iterations
16	21845
20	349525
24	5592405
28	89478485

Table 1: Number of disks vs Number of iterations required to solve

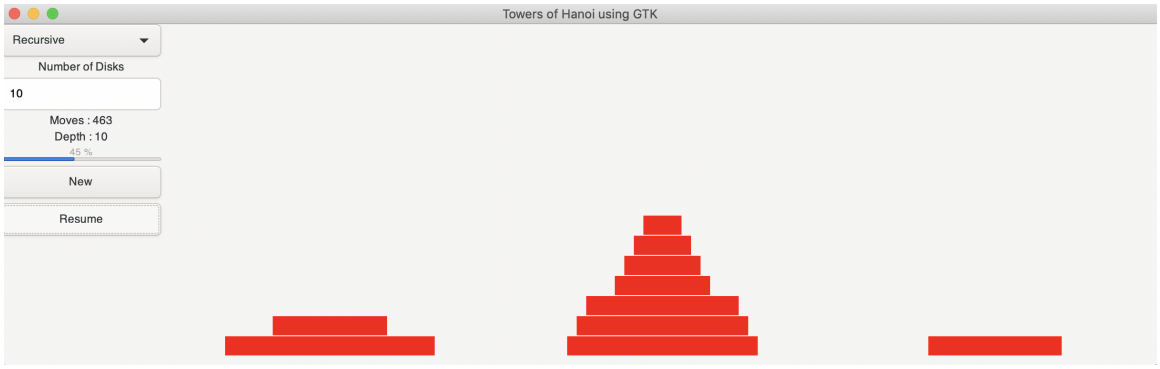
Algorithm Visualization

The impact of the performance implications would be readily apparent if the execution is visualized. For example, if the moves of the disks are presented graphically, and every move is presented in a graphical interface, the sheer number of moves and the time it takes would be clear. By varying the number of disks, we can immediately appreciate the relation. The ability to pause/resume the progress proves to be highly educational

Animation - fyne graphical library



Animation - gtk3 library



Implementation

Example Project

An example project has been developed to illustrate the concepts - with the following specifications:

```
usage: hanoi <number of disks> [iterative] [graphic]
iterative - use the iterative solution. graphic - present the solution graphically
```

An implementation in **go** language using **fyne** (`https://fyne.io/`) library for gui has been developed and is available for exploration and enhancements. The project also incorporates an alternative implementation using the **gtk3** framework (`https://github.com/gotk3/gotk3`).

```
https://sourceforge.net/p/gohanoi/code/ci/master/tree/
```